



# Substrate 区块链应用开发

Runtime 数据存储的设计

---

孙凯超

kaichao@parity.io

获取帮助: <https://substrate.io>

# 内容

---

- 区块链存储的不同点和约束
- Substrate 存储单元的类型
- 存储的初始化
- 最佳实践

# 区块链存储的不同点

---

区块链应用通常几个特点，

- 开源可审查，对等节点，引入延迟和随机来达到共识
- 链式、增量地存储数据

区块链应用的节点软件依赖高效的键值对数据库：

- LevelDB
- RocksDB

# 区块链存储的约束

---

区块链作为业务的载体，存储相关的限制有：

- 大文件直接存储在链上的成本很高；
- 链式的区块存储结构不利于对历史数据的索引；
- 另外一个约束是，在进行数值运算时不能使用浮点数。

# Substrate 存储单元的类型

---

## 开发链上存储单元的特点：

- Rust原生数据类型的子集，定义在核心库和alloc库中
- 原生类型构成的映射类型
- 满足一定的编解码条件

单值

简单映射

双键映射

## 回顾 decl\_storage 宏

---

/// The pallet's configuration trait.

```
pub trait Trait: system::Trait {  
    /// The overarching event type.  
    type Event: From<Event<Self>> + Into<<Self as system::Trait>::Event>;  
}
```

// This pallet's storage items.

```
decl_storage! {  
    trait Store for Module<T: Trait> as TemplateModule {  
        Something get(fn something): Option<u32>;  
    }  
}
```

# 单值类型

存储某种单一类型的值，如布尔，数值，枚举，结构体等：

- 数值：`u8, i8, u32, i32, u64, i64, u128, i128`
- 大整数：`U128, U256, U512`
- 布尔：`bool`
- 集合：`Vec<T>, BTreeMap, BTreeSet`
- 定点小数：`Percent, Permill, Perbill`
- 定长哈希：`H128, H256, H512`
- 其它复杂类型：`Option<T>, tuple, enum, struct`
- 内置自定义类型：`Moment, AccountId`

# 单值类型

---

数值类型 `u8, i8, u32, i32, u64, i64, u128, i128` 的定义：

```
decl_storage! {  
    trait Store for Module<T: Trait> as DataTypeModule {  
        // store unsigned integer, init to zero if not set  
        MyUnsignedNumber get(fn unsigned_number): u8 = 10;  
        // also init to zero, can store negative number  
        MySignedNumber get(fn signed_number): i8;  
    }  
}
```



# 单值类型

---

数值类型 `u8, i8, u32, i32, u64, i64, u128, i128` 的使用：

- 增：`MyUnsignedNumber::put(number);`
- 查：`MyUnsignedNumber::get();`
- 改：`MyUnsignedNumber::mutate(|v| v + 1);`
- 删：`MyUnsignedNumber::kill();`

更多API, 请参考文档 Trait [frame\\_support::storage::StorageValue](#)

# 单值类型

---

数值类型 `u8, i8, u32, i32, u64, i64, u128, i128` 的安全操作：

- 返回Result类型：`checked_add`, `checked_sub`, `checked_mul`, `checked_div`

```
// fail the transaction if error  
my_unsigned_num.checked_add(10)?;
```

- 溢出返回饱和值：`saturating_add`, `saturating_sub`, `saturating_mul`  
// result is 255 for u8  
`my_unsigned_num.saturating_add(10000);`

# 单值类型

---

大整数 `U256, U512` 类型定义：

```
use sp_core::U256;
decl_storage! {
    trait Store for Module<T: Trait> as DataTypeModule {
        // init to 0
        MyBigInteger get(fn my_big_integer): U256;
    }
}
```

# 单值类型

大整数 `U256, U512` 类型定义：

```
use sp_core::U256;
decl_storage! {
    trait Store for Module<T: Trait> as DataTypeModule {
        // init to 0
        MyBigInteger get(fn my_big_integer): U256;
    }
}
```

操作：`checked_add, overflowing_mul...`

更多API, 参考文档 [sp\\_core::U256](#)

# 单值类型

---

**bool** 类型定义：

```
decl_storage! {  
    trait Store for Module<T: Trait> as DataTypeModule {  
        // init to false, store boolean value  
        MyBool get(fn my_bool): bool;  
    }  
}
```

# 单值类型

---

`Vec<T>` 类型定义：

```
use sp_std::prelude::*;
```

```
decl_storage! {  
    trait Store for Module<T: Trait> as DataTypeModule {  
        // default to 0x00  
        MyString get(fn my_string): Vec<u8>;  
    }  
}
```

# 单值类型

`Vec<T>` 类型定义：

```
use sp_std::prelude::*;
```

```
decl_storage! {  
    trait Store for Module<T: Trait> as DataTypeModule {  
        // default to 0x00  
        MyString get(fn my_string): Vec<u8>;  
    }  
}
```

操作：`push, pop, iter...` [Vec结构体](#)

# 单值类型

---

Percent, Permill, Perbill 类型定义：

```
use sr_primitives::Permill;
decl_storage! {
    trait Store for Module<T: Trait> as DataTypeModule {
        // fix point number, default to 0
        MyPermill get(fn my_permill): Permill;
    }
}
```



# 单值类型

---

Percent, Permill, Perbill 类型操作：

- 构造
  - `Permill::from_percent(value);`
  - `Permill::from_parts(value);`
  - `Permill::from_rational_approximation(p, q);`
- 计算
  - `permill_one.saturating_mul(permill_two);`
  - `my_permill * 20000 as u32`

API文档 [sp\\_arithmetic::Permill](#)

# 单值类型

---

**Moment** 时间类型定义：

```
pub trait Trait: system::Trait + timestamp::Trait {}  
decl_storage! {  
    trait Store for Module<T: Trait> as DataTypeModule {  
        // Moment is type alias of u64  
        MyTime get(fn my_time): T::Moment;  
    }  
}
```

获取链上时间：`<timestamp::Module<T>>::get();`

# 单值类型

---

AccountId 账户类型定义：

```
decl_storage! {  
    trait Store for Module<T: Trait> as DataTypeModule {  
        MyAccountId get(fn my_account_id): T::AccountId;  
    }  
}
```

获取AccountId: `let sender = ensure_signed(origin)?;`

# 单值类型

---

**struct** 类型定义：

```
#[derive(Clone, Encode, Decode, Eq, PartialEq, Debug, Default)]
pub struct People {
    name: Vec<u8>,
    age: u8,
}

decl_storage! {
    trait Store for Module<T: Trait> as DataTypeModule {
        MyStruct get(fn my_struct): People;
    }
}
```

# 单值类型

---

enum 类似, 还需要实现Default接口

<https://github.com/kaichaosun/play-substrate/blob/master/pallets/datatype/src/lib.rs#L32>

# 简单映射类型

---

**map** 类型，用来保存键值对，单值类型都可以用作key或者value，定义：

```
decl_storage! {  
    trait Store for Module<T: Trait> as DataTypeModule {  
        MyMap get(fn my_map): map hasher(twox_64_concat) u8  
=> Vec<u8>;  
    }  
}
```

# 简单映射类型

---

`map` 类型，用来保存键值对，单值类型都可以用作key或者value，定义：

```
decl_storage! {  
    trait Store for Module<T: Trait> as DataTypeModule {  
        MyMap get(fn my_map): map hasher(twox_64_concat) u8  
=> Vec<u8>;  
    }  
}
```

hasher: `blake2_128_concat`, `twox_64_concat`, `identity`

# 简单映射类型

---

**map** 类型，用来保存键值对，单值类型都可以用作key或者value，用法：

- 插入一个元素：`MyMap::insert(key, value);`
- 通过key获取value：`MyMap::get(key);`
- 删除某个key对应的元素：`MyMap::remove(key);`
- 覆盖或者修改某个key对应的元素
  - `MyMap::insert(key, new_value);`
  - `MyMap::mutate(key, |old_value| old_value+1);`

API 文档：[Trait frame\\_support::storage::StorageMap](#)

[Trait frame\\_support::storage::IterableStorageMap](#)



# 双键映射类型

---

`double_map` 类型，使用两个key来索引value，用于快速删除key1对应的任意记录，也可以遍历key1对应的所有记录，定义：

```
decl_storage! {  
    trait Store for Module<T: Trait> as DataTypeModule {  
        MyDoubleMap get(fn my_double_map): double_map hasher  
        (blake2_128_concat) T::AccountId, hasher(blake2_128_concat) u32 =>  
        Vec<u8>;  
    }  
}
```

# 双键映射类型

---

`double_map` 类型，使用两个key来索引value，用于快速删除key1对应的任意记录，也可以遍历key1对应的所有记录，定义：

- 插入一个元素：`MyDoubleMap::<T>::insert(key1, key2, value);`
- 获取某一元素：`MyDoubleMap::<T>::get(key1, key2);`
- 删除某一元素：`MyDoubleMap::<T>::remove(key1, key2);`
- 删除 key1 对应的所有元素  
：`MyDoubleMap::<T>::remove_prefix(key1);`

API文档 [frame\\_support::storage::StorageDoubleMap](#)

[Trait frame\\_support::storage::IterableStorageDoubleMap](#)

# 存储的初始化

---

创世区块的数据初始化，有三种方式：

- `config()`
- `build(clousure)`
- `add_extra_genesis { ... }`

# 存储的初始化

---

演示:

<https://github.com/kaichaosun/play-substrate/blob/master/pallets/genesis-config/src/lib.rs>

# 最佳实践

---

- 最小化链上存储
  - 哈希值
  - 设置列表容量
- Verify First, Write Last

## 其它Tips

---

- 可以通过pub关键字设置存储单元的可见范围
- 可以手动设置默认值，如：
  - `MyUnsignedNumber get(fn unsigned_number): u8 = 10;`
- 在frame目录下查找对应的最新用法
- [decl\\_storage](#) 宏的说明文档

# Questions?

---

官网文档: [substrate.io](https://substrate.io)

知乎专栏: [parity.link/zhihu](https://parity.link/zhihu)

[kaichao@parity.io](mailto:kaichao@parity.io)

weixin415148673